1. 手动提取特征的缺点

- 挖掘高质量的交互特征需要非常专业的领域知识并且需要做大量尝试，耗费时间和精力。
- 在大型推荐系统中，原生特征非常庞大，手动挖掘交叉特征几乎不可能。
- 挖掘不出肉眼不可见的交叉特征。

2. FM系列模型

FM模型：提取隐向量然后做内积的形式来提取交叉特征，扩展的FM模型更是可以提取随机的高维特征（DeepFM），**缺点：会学习所有交叉特征，其中肯定会包含无用的交叉组合，这些组合会引入噪音降低模型的表现。（前期的特征选择很重要）**

针对FM、FFM模型的缺点，随后又出现了几种针对改进的模型：

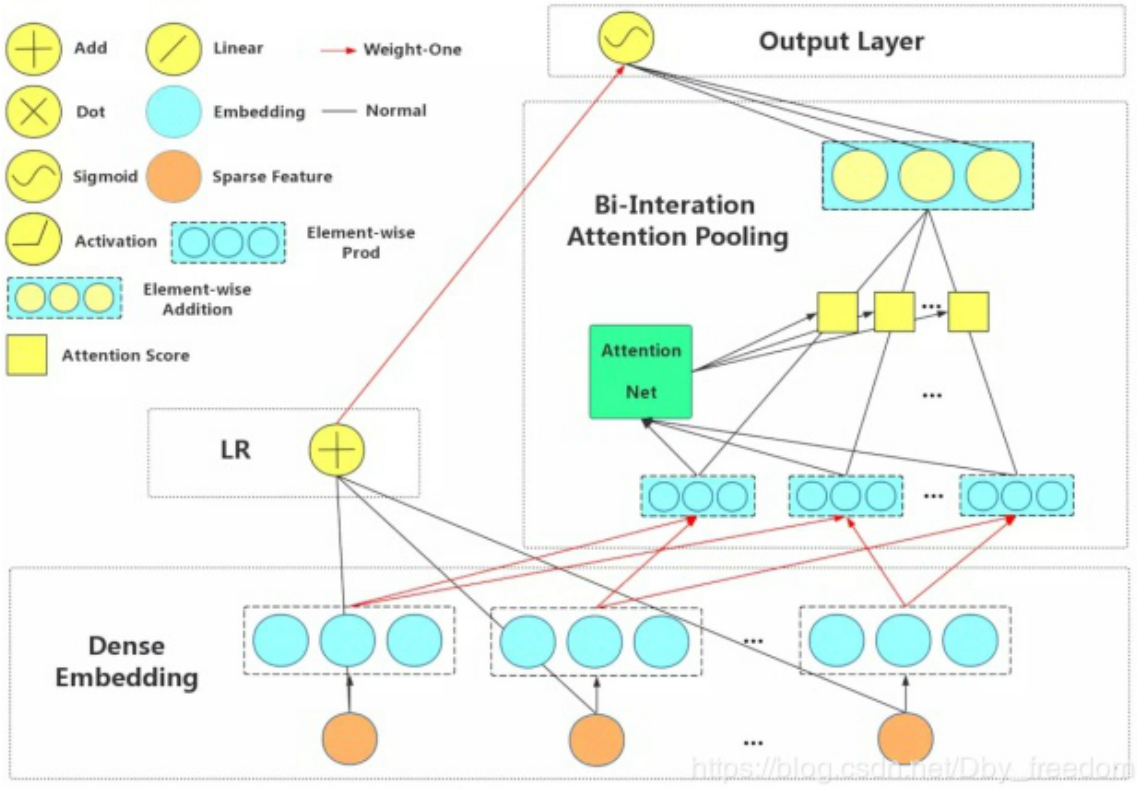
2.1 FwFM

FFM 算法按照 field 对 latent vector 进行区分，从而提升模型的效果。但是 **FFM 算法没有区分不同特征交叉的重要性，此模型针对不同特征交叉赋予不同的权重**，从而达到更精细的计算交叉特征的目的。

$$\mathrm{y}(\mathbf{x}) := w_0 + \sum_{i=1}^{n} x_i w_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} < \mathbf{v}_i, \mathbf{v}_j > x_i x_j r_{\mathrm{F}(i),\mathrm{F}(j)}$$

其中，$r_{F(i),F(j)}$ 表示 field $F(i), F(j)$交叉特征的重要性。

2.2 AFM



AFM 和 FwFM 类似，目标是希望对不同交叉特征采用不同的权重，从而减少引入噪声提升模型的性能。

如图所示，**AFM在embedding后，先让 f 个field的特征做了bit-wise product后，得到** $f \times (f-1)/2$ 交叉项，然后**AFM 引入了一个attention net**，认为这些交叉特征项对每个结果贡献是不同的，例如 x_i, x_j 的权重重要度，用 a_{ij} 来表示。从这个角度来看，其实 **AFM 就是个加权累加的过程**。

1. Attention-based Pooling Layer

$$a'_{ij} = \mathbf{h}^{\mathrm{T}} \operatorname{ReLU}\left(\mathbf{w}\left(\mathbf{v}_i \odot \mathbf{v}_j\right) x_i x_j + \mathbf{b}\right)$$

$$a_{ij} = \frac{\exp\left(a'_{ij}\right)}{\sum_{(i,j)\in \Re x} \exp\left(a'_{ij}\right)}$$

> 先计算伪权重，再进行归一化为真权重2. AFM模型结构

2. AFM 模型结构

$$y(x) = w_0 + \sum_{i=1}^{n} x_i w_i + \mathbf{p}^{\mathrm{T}} \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{ij}\left(\mathbf{v}_i \odot \mathbf{v}_j\right) x_i x_j$$

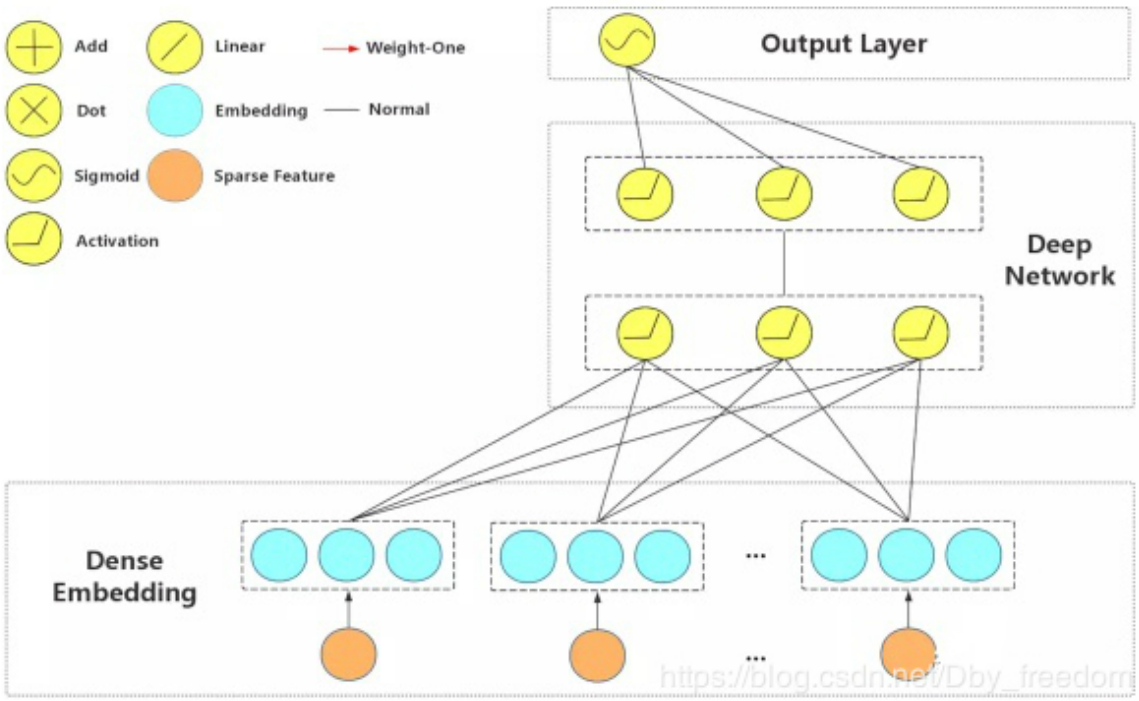
其中，h,w,p,b为模型参数。

以上均为低阶特征交互模型，并不能实现高阶的特征交互！

3. DNN模型

- **FNN：** 在 DNN 结构之间使用了 field embedding（串联），和 DeepFM 区别开来，DeepFM 是并联方式。
- **PNN：** 在 embedding layer 和 DNN Input之间插入一层 product layer，不依赖于 pre-trained FM。

3.1 FNN



FNN 层用 FM 初始化，即 FM 模型作为 pre-training，得到每个特征对应一个偏置项 w_i 和一个 k 维向量 v_i。然后参数向量随着训练不断学习调整。优势自然就是 FNN 训练开销低，可以更快达到收敛(初始化 embedding 为 FM 模型的 pre-training)

$$z_i = \mathbf{W}_0^i \cdot x\left[\text{start}_i : \text{ end }_i\right] = \left(w_i, v_i^1, v_i^2, \ldots, v_i^K\right)$$

$$\mathbf{I}_1 = \tanh(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1)$$

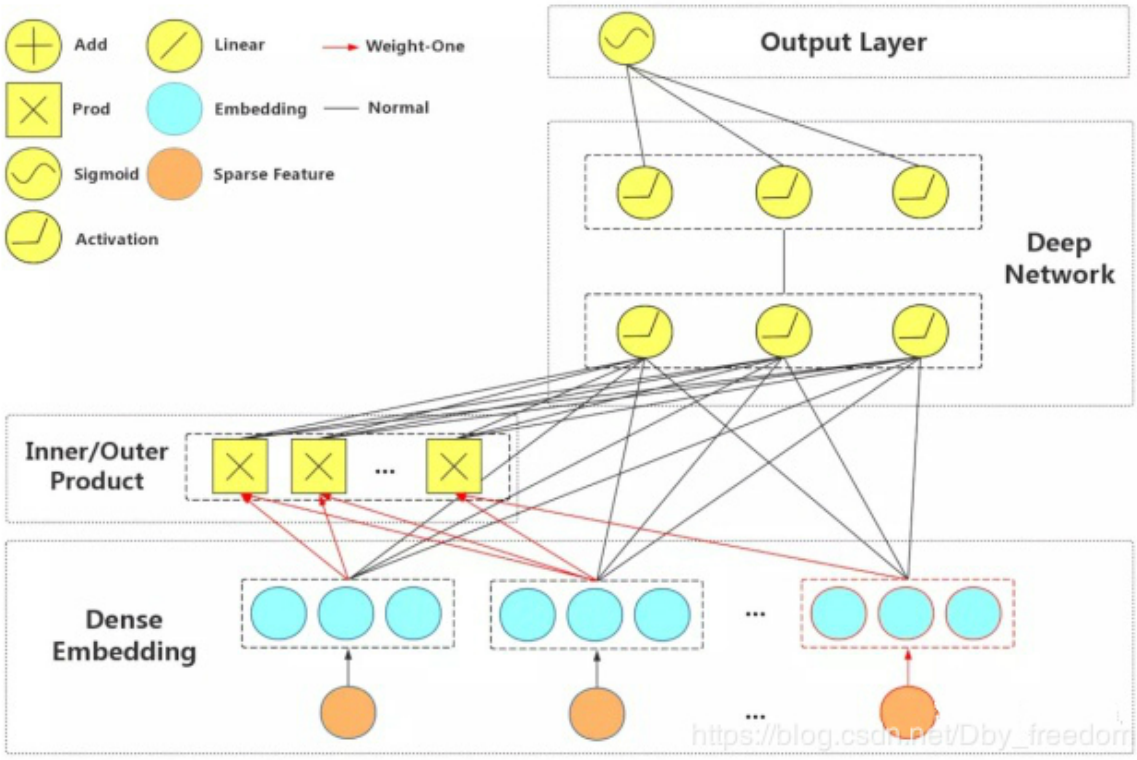
$$\mathrm{I}_2 = \tanh(\mathbf{W}_2 \mathbf{I}_1 + \mathbf{b}_2)$$

$$\hat{y} = \text{sigmoid}(\mathbf{W}_3 \mathbf{I}_2 + \mathbf{b}_3)$$

损失函数（最小交叉熵）为：

$$L(y, \hat{y}) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

3.2 PNN



FNN 实际上是对特征 embedding 之后进行 concatenate，再接 MLP，虽然使用激活函数(tanh)增加非线性，实际上是对特征进行了加权组合，**PNN 与 FNN 区别在于 PNN 中间多了一层 Product Layer 层。其中 z 为embedding 层的线性部分，p 为 embedding 层的特征交叉部分**，其他与 FNN 算法结构相同。

$$\begin{aligned}
l_z &= \left(l_z^1, l_z^2, \ldots, l_z^n \ldots, l_z^{D_1}\right), l_z^n = W_z^n \odot z \\
l_p &= \left(l_p^1, l_p^2, \ldots, l_p^n \ldots, l_p^{D_1}\right), l_p^n = W_p^n \odot p \\
l_1 &= relu\left(l_z + l_p + b_1\right); l_2 = relu\left(W_2 l_1 + b_2\right) \\
\tilde{y} &= \sigma\left(W_3 l_2 + b_3\right)
\end{aligned}$$

product layer 分为两部分，其中 **z** 代表线性信号向量，而 **p** 代表二次信号向量；

1. Inner Product-based Neural Network：
   $g(f_i, f_j) =< f_i, f_j >$，内积表示特征交叉，类似于“且”，$f_i$ 为embedding向量。
2. Outer Product-based Neural Network
   $g(f_i, f_j) = f_i f_j^T$，矩阵乘法表示特征交叉，类似于“和”的关系。

FNN 和 PNN 缺点都是忽略了低维交叉特征，Wide&Deep 和 DeepFM 模型通过混合架构解决了这个问题，但是同样存在缺点：

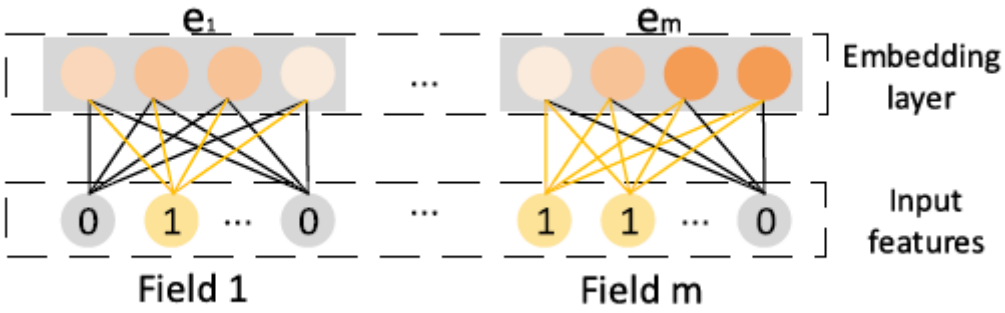
- 它们学习到的高维特征为 implicit 隐式的，也就是说没有推论得出最终学习出来的交叉特征到底是多少维的；
- DNN 部分是在 bit-wise 的层面下进行学习的，经典的 FM 框架是在 vector-wise 层面下学习。

> **Bit-wise VS vector-wise**
> 假设隐向量的维度为3维，如果两个特征(对应的向量分别为 (a1,b1,c1) 和 (a2,b2,c2) 的话）在进行交互时，交互的形式类似于 f(w1 * a1 * a2,w2 * b1 * b2 ,w3 * c1 * c2) 的话，此时我们认为特征交互是发生在元素级（bit-wise）上。如果特征交互形式类似于 f(w * (a1 * a2 ,b1 * b2,c1 * c2))的话，那么我们认为特征交互是发生在特征向量级（vector-wise）。

> **Explicitly VS implicitly**
> 显式的特征交互和隐式的特征交互。以两个特征为例 x_i和 x_j，在经过一系列变换后，我们可以表示成 $w_{ij} * (x_i * x_j)$ 的形式，就可以认为是显式特征交互，否则的话，是隐式的特征交互。

4. Embedding layer

在推荐系统中，**输入的原始特征稀疏，维数大，没有明显的时空相关性**，在原始特征输入上应用embedding层，将其压缩为低维、密集的实值向量。



输入的每个 **feature** 长度可能不同，但是经过 **embedding layer**，维数均保持为**k**。

5. 隐式的高阶交互

FNN、Deep Crossing 以及 Wide&Deep 的深层部分利用 field embedding e 上的前馈神经网络来学习高阶特征交互，

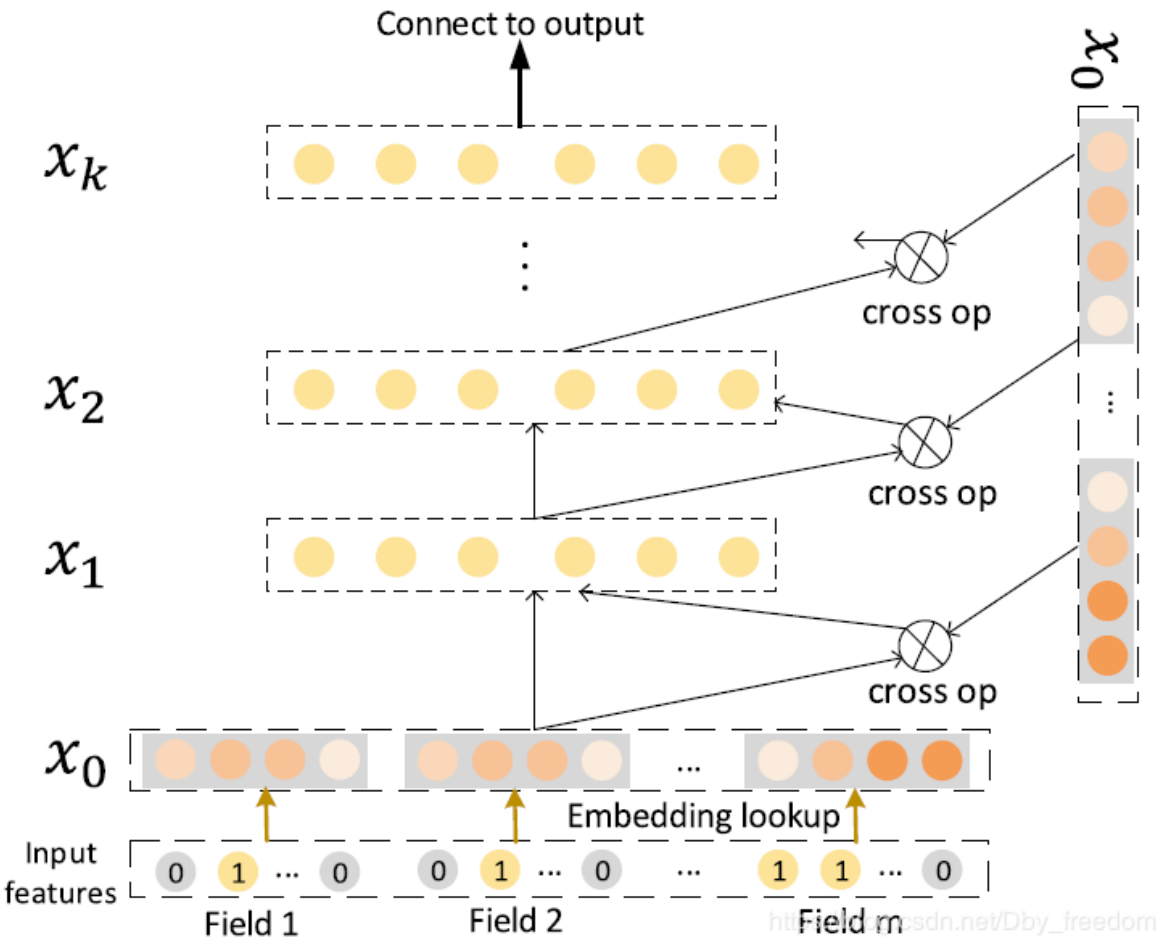
$$\mathbf{x}^1 = \sigma\left(\mathbf{W}^{(1)}\mathbf{e} + \mathbf{b}^1\right)$$

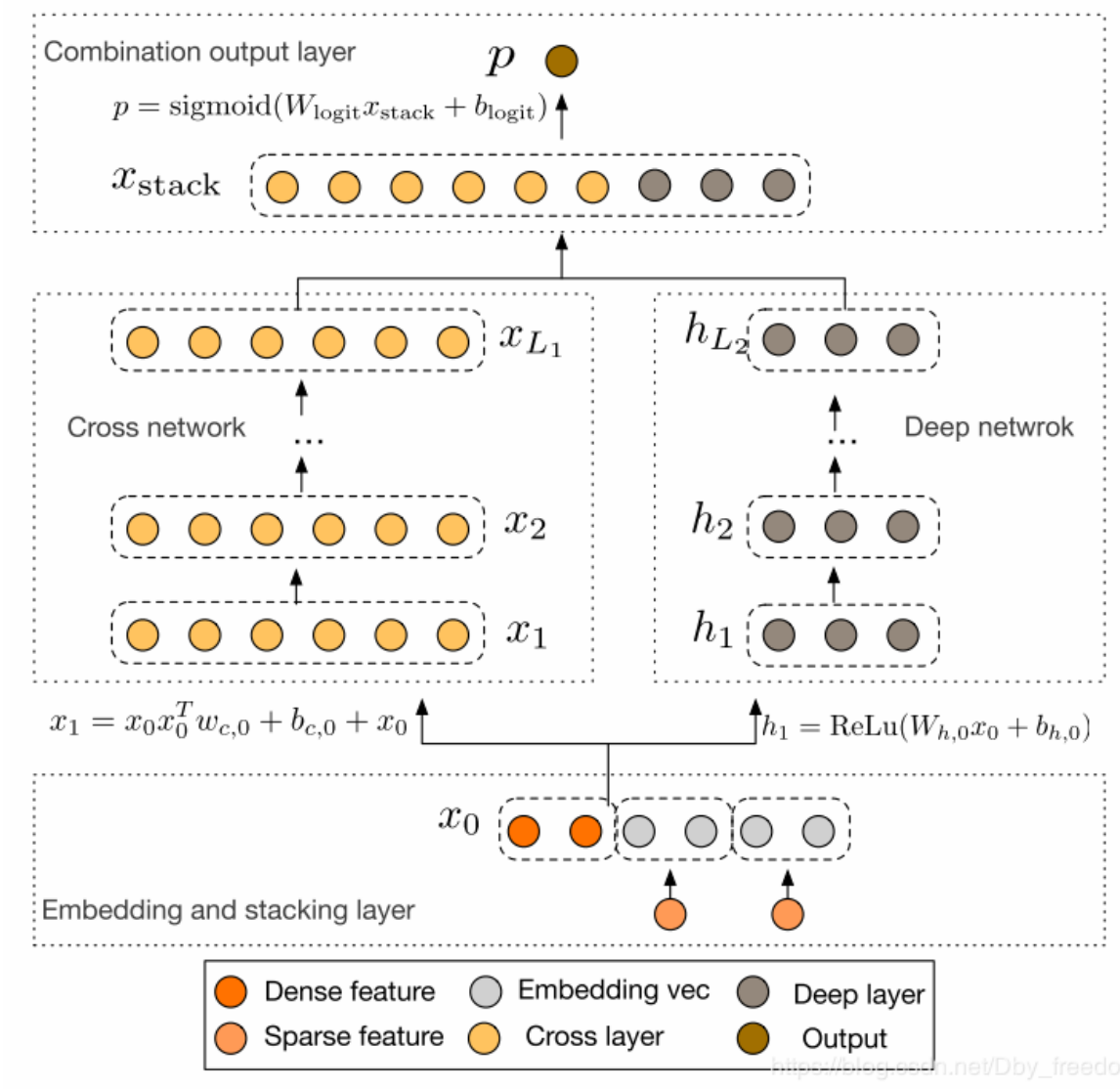
$$\mathbf{x}^k = \sigma\left(\mathbf{W}^{(k)}\mathbf{x}^{(k-1)} + \mathbf{b}^k\right)$$

PNN 和 DeepFM 除了在嵌入向量 e 应用 DNN 外，还引入了双向交互层(FM)，PNN 和 DeepFM 之间的主要区别在于：PNN 将 product layer 的输出连接到 DNN，而 DeepFM 将 FM 层直接连接到输出单元。（一个是串联一个是并联）之所以称为隐式的高阶交互，是因为这两个模型得到 DNN 部分并不能推断出具体学到了多少阶的交叉特征。

6. 显示的高阶交互

6.1 Cross-Network结构





Cross-Network 的目标在于明确地模拟高阶特征交互。与传统的全连接网络不同，隐藏层通过以下交叉操作计算：

- cross layer: $x_{l+1} = x_0 x_l^T w_l + b_l + x_l = f(x_l, w_l, b_l) + x_l$
- deep layer: $h_{l+1} = f(w_l h_l + b_l)$，$f(\cdot)$ 为 relu 激活函数。
- combination output layer：$p = \sigma\left(\left[x_L^T, h_{L_2}^T\right] W_{logits}\right)$，将经过 cross layer 的输出 x 和经过 deep layer 的输出 h 进行 concat 得到最终的特征向量。σ 为 sigmoid 函数。

缺点：（1）Cross-Network的输出以特殊形式限制，**每个隐藏层是 x_0 标量的倍数**；（2）特征交互以 bit-wise 方式进行。

对于 cross layer 层，有 $x_{l+1} = x_0 x_l^T w_l + b_l + x_l = f(x_l, w_l, b_l) + x_l$ ，去掉偏置项 b_l 以后有：

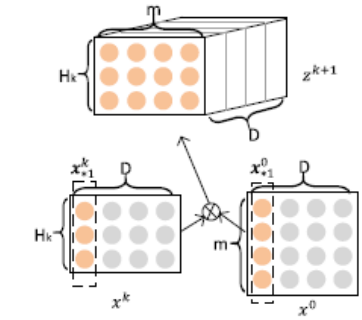
$$x_{i+1} = x_0 x_i^T w_{i+1} + x_i = x_0 \left(\left(\alpha^i x_0\right)^T w_{i+1}\right) + \alpha^i x_0 = \alpha^{i+1} x_0$$

α 是一个标量，多层之后 x_{i+1} 仍是 x_0 与表里的乘积，特征交叉只是 bit-wise 级。

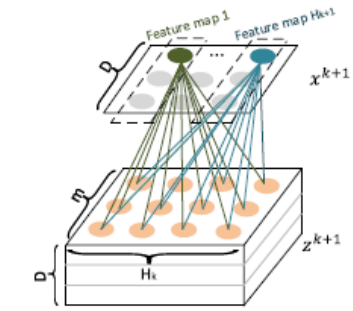
6.2 CIN (Compressed Interaction Network) 与 xDeepFM 模型

- 相互作用在 vector-wise 层面，而不是在 bit-wise 上；

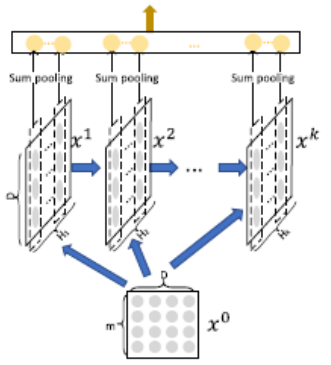
- 显式度量高阶特征相互作用；
- 网络的复杂性不会随着交互程度指数级增长；



(a) Outer products along each dimension for feature interactions. The tensor Z^{k+1} is an intermediate result for further learning.

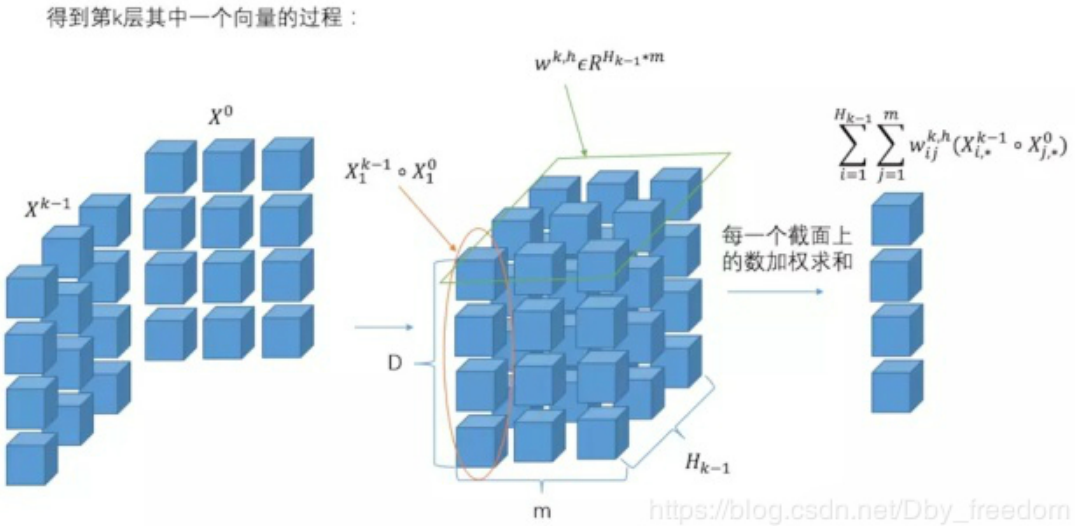


(b) The k-th layer of CIN. It compresses the intermediate tensor Z^{k+1} to H_{k+1} embedding vectors (aslo known as *feature maps*).



(c) An overview of the CIN architecture.

1. 如 (a) 所示，根据原始特征矩阵 x_0 $(H_k \times D)$ 和前一层的输出 $x^k(m \times D)$，得到中间结果 z^{k+1} $(H_k \times m \times D)$；
2. 图 (b) 为 feature map 过程，生成的中间结果 z^{k+1}，利用 H_{k+1} 个尺寸大小为 $m \times H_k$ 的卷积核得到下一层的隐层状态。具体的卷积计算细节与 CNN 不同，CIN 中一个神经元相关的接受域是垂直于特征维度 D 的整个平面，而 CNN 中的接受域是当前神经元周围的局部小范围，因此 CIN 经过卷积操作得到的特征图是一个向量，而不是一个矩阵。



$$X_{h,*}^k = \sum_{i=1}^{H_{k-1}} \sum_{j=1}^{m} W_{ij}^{k,h} \left(X_{i,*}^{k-1} \cdot X_{j,*}^0 \right)$$

w 类似于 CNN 中的 filter， $X_{h,*}^k$ 就是一个 feature map。

$p_i^k = \sum_{j-1}^{D} X_{i,j}^k$，其中，$i \in [1, H_k]$，这样得到一个pooling vector：$p^k = \left[p_1^k, p_2^k, \ldots, p_{H_k}^k \right]$，$p^+ = \left[p^1, p^2, \ldots, p^T \right]$，T 表示网络的深度。

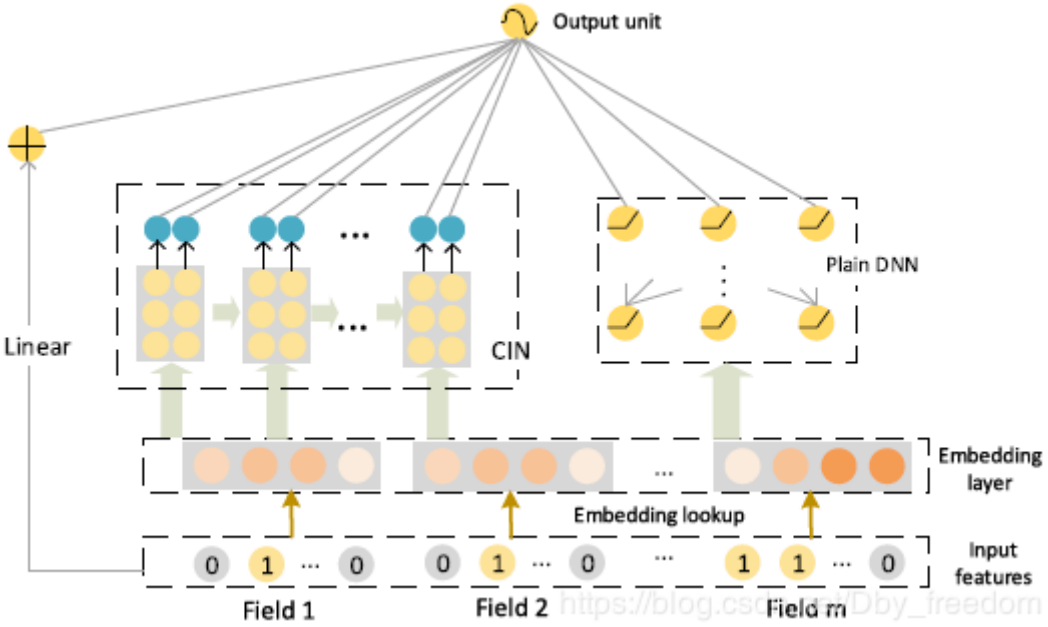
$$\tilde{y} = \sigma \left(w_{\text{linear}}^T a + w_{dnn}^T x_{dnn}^K + w_{cin}^T p^+ + b \right)$$

x_{dnn}^k, p^+ 分别是 DNN 和 CIN 的输出。

3. 对于图（c），每一层的隐状态 x^{k+1}，将每个(第 k 层有 H_k 个)特征向量求和，（sum pooling），最后将每一层结果拼接起来，作为最后的输出。

CIN 实现了输出单元可以得到不用阶数的特征交互模式，此外 CIN 的计算过程与机构和 RNN 类似，对于每一层的隐状态都是根据前一层的隐状态和额外的输入数据计算得到。

将 CIN 和 DNN 结合起来，得到 xDeepFM



一方面包括低阶和高阶特征交互；另一方面，既包括隐式的功能交互，也包括显式的功能交互；分别是DNN和CIN的输出。

损失函数（log loss）：

$$\Gamma = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log \tilde{y}_i + (1-y_i)\log(1-\tilde{y}_i)) + \lambda\Theta$$

θ 为 l1 和 l2 正则化。

> 暂记于此，后续继续补充。

参考文献：

[1] 博客：https://blog.csdn.net/yfreedomliTHU/article/details/91386734blog.csdn.net

[2] 综述：机器学习在CTR中的应用

[3] xDeepFM